IA311-Final Report
Skills Similarity Measurement To Enrich An Ontology

Supervisor: M. Mohammad Fallaha
Referent : Mrs. Clavel

Aurélien Chambon
Vathana Thy

February 2021

Context

This work has been done during our final year at Télécom SudParis and Télécom Paris, under the supervision of M. Mohammad Fallaha from September 2020 to February 2021. Due to the pandemic, it was rather complicated to work physically at xLearn (the start-up of M. Fallaha who is our supervisor). Therefore, the physical meeting needed to be transformed into the form of weekly video conference meetings with the supervisor.

During this project, we were interested in building a machine learning model based on word embedding to standardize the skills of a candidate by finding the most similar skill from the skills ontology provided by xLearn.

This work is done by two students : Aurélien Chambon (Télécom SudParis) and Vathana Thy (Télécom Paris). We have been working together all along this project, but for the sake of clarity and facility in evaluating our individual contribution, we are going to denote each section with (A) and (V) for Aurélien and Vathana respectively to indicate who contribute the most for the section.

Introduction

Word embedding is a technique to represent words into vectors in a specific way. As you can see in the figure 1, this technique represents individual words by real-valued vectors whose values are learned in a way that words having the same meaning have similar values. In doing so, it embeds the semantic link between words. Word2Vec is an algorithm that can be used to learn a word embedding model from data. We then use this model in order to match job seeker's resume to job descriptions through distances between candidate skills and skills requirement. Our report is organized as below:

- In section 1, we will do a brief presentation of xLearn, the start-up with whom we achieved this project.
- Section 2 is dedicated to the database pre-processing. We will talk briefly about the database overview before going into detail about all the operations in order to clean the database.
- In section 3, we will talk about modeling. This section describes basically the algorithms being used in order to learn the word representation from the database.
- Section 4 is about the tool or python packages that we used to perform the skills extraction from text such as job seekers' resumes which will be used to match with job description.
- In section 5, we will explain our methods based on these packages or tools, described in section 4, to extract skills and to compute the similarities between a resume and a job description.
- Finally, in section 6, we will talk about the implementation of our project into an API and a web interface.

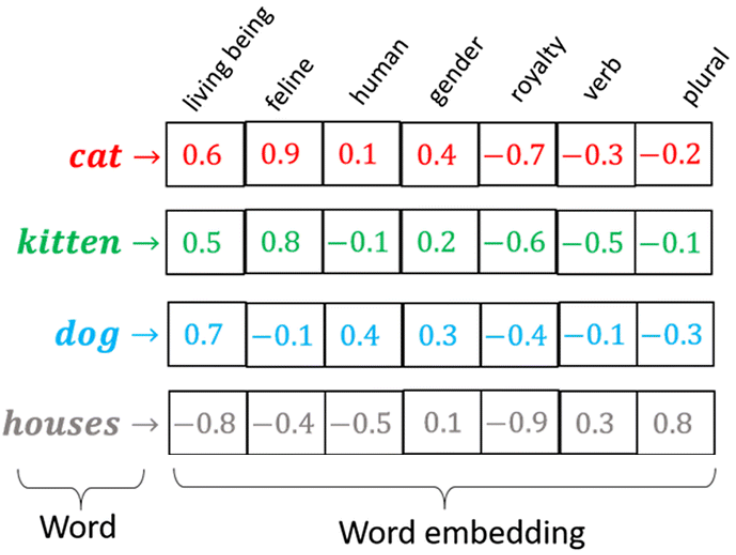

Figure 1: Visualisation of word embeddings by Hariom Gautam on his medium

Contents

1 Presentation of xLearn

xLearn is a start-up founded in Télécom Paris Entrepreneurs Incubator. Its main activities include developing tools that enable client companies to support their employees in career advancement by offering them opportunities, positions and projects which correspond better to their expectations and skills. In order to do so, they provide Human Resources (HR) with consultations and tools to track the employees performances. Our work in this project is part of a feature to help HR in their resume selection.

2 Database pre-processing

2.1 Database overview

Our resource for this project is a database containing skills in two languages: English and French, listed by *name*, *id*, and a list of 5 to 9 skills, which appear the most frequently in users profiles with the skill (from the column *name*), named *topSkills*. The skills in *topSkills* are in a specific format, which is a dictionary containing the name of the topskill and the memberCount, the number of users profiles having both the topskill and the skill in their description.

For example, the skill *Computer Vision* has *C++, Matlab, Machine Learning, Python, C, Algorithms, Image Processing, Java, Programming, Software Development* in its *topSkills* as they often appear together in users profiles.

In this database, we can find around 34000 rows which may contain duplicates, null values and rows referring to the same skill such as **Java** and **Java programming** for instance. Furthermore, as explained previously, skills listed in *topSkills* don't have their *id* in their dictionary, which we have to add in order to link, via the id, the "skill object" to each topskill in *topSkills*. Here is a look at the first five rows of our original dataset :

	_id	name	topSkills
0	5b057b59b5d2691588302c95	A + Certified	[{'name': 'Certification Network+ ', 'memberCo...
1	5b057b59b5d2691588302c96	A&d	[{'name': 'Pétrole ', 'memberCount': 694}, {'n...
2	5b057b59b5d2691588302c97	A&e	[{'name': 'Services de santé ', 'memberCount':...
3	5b057b59b5d2691588302c98	A&h	NaN
4	5b057b59b5d2691588302c99	A&p	[{'name': 'Aviation ', 'memberCount': 6175}, {...
...	...	...	...

Figure 2: Database overview

2.2 Data cleaning and pre-processing

2.2.1 Handling null values

An important part of every data science project is to have data with quality. Hence, we spend a lot of time cleaning and pre-processing. Our first operation has been to delete every row

where *name* has a null value but to keep those where *topSkills* are null as we can still use the relation between *name* and *id* of the skill row. This will be useful to add *id* to our elements in *topSkills*.

2.2.2 Language detection (V)

Before proceeding to another pre-processing step, we need to handle the language of the database. After testing, we found more than 95% of the data are in English. Due to computation time, we are first interested in the case of transforming the whole database into English. We can optimize this process by translating only the words detected as French into English.

```
There are 13289 French skills (3.903489318203848%) and 327150 English skills (96.09651068179616%).
```

Figure 3: French and English skills proportion

One of the main challenges here is to find the most accurate way possible to detect language of skills. Relying on only one classifier is intuitively not the best solution as we risk of having undesired result. Therefore, we think of having many classifiers and doing a majority vote to determine the language of the skills.

Unfortunately, we are able to use only one classifier (*Langid*), which is reasonable to be used in terms of execution time and result, in order to classify the language of the skills, since other classifiers than *Langid* (NLTK, Googletrans, Pycld3, Langdetect, Chardet, GuessLanguage) don't allow user to specify the desired language to be detected (English and French in our case). These classifiers either have very poor and unusable performance or do not work at all. The accuracy of *Langid* is not 100%. However, it is good enough to be used (cf. figure 4).

	skill	langid	nltk	googletrans	pycld3	langdetect
0	Aviation	en	lms	en	mg	en
1	Maintenance avions	fr	fra	en	ny	fr
2	Compagnie aérienne	fr	fra	en	fr	fr
3	Aérospatiale	fr	por	en	fr	fr
4	Navigabilité	fr	lld	en	fr	fr
5	Aviation commerciale	fr	lms	en	en	it
6	Sécurité en vol	fr	fra	en	fr	fr
7	Avionique	fr	lnc	en	la	fr
8	Aéronautique	fr	fra	en	fr	fr
9	Systèmes d'avion	fr	fra	en	fr	fr

Figure 4: Comparison of different classifier's prediction

2.2.3 Skills translation (V)

For each skill detected as a French word, we translate it into English and put the skill translated back into the database. We have tried two different tools for translation (GoogleTrans and Google API) and Google API appears to do its tasks well while GoogleTrans is very unstable or does not provide a good translation.

After this step, we have our whole database in English. We then continue to the following step of pre-processing.

2.2.4 Pre-processing : tokenization and stemming (A)

We then had to pre-process our data. Usually, we start by removing accents, superfluous spaces, stopwords and uppercases. We will call this transformation *clean names*. Next, we tokenize sentences, add lemmatization and stemming of the skills' name. However, we decided to focus on stemming rather than lemmatization. Indeed, as we handle english skills and using only skill names (not sentences), we don't need the syntactic analysis from lemmatization. Here is a simple example of our preprocessing so far which we called *tokenized clean name* :

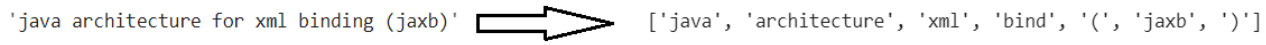

Figure 5: Example of skill tokenization

2.2.5 Searching for duplicates (A)

Let's consider the skill **Javascript (JS)** which is the same conceptual skill as **Javascript** and **JS**. However these three skills have a different id in our dataset. We had to handle such duplicates.

Firstly, we keep only one row where skills' *clean name* are the same. For example: **Data Marketing** and **data marketing** both have the same *clean name* so we should keep only one.

Then, we use tokenized skill in order to find which skill has another skill within its *tokenized clean name*. If such related skills are found, we then calculate the proportion of common skills in their *topSkills* and delete one of them if this proportion is above 90%.

However, we don't want to lose any information, so we added a new column *synonyms* in which we put the deleted duplicate *name*. For example, if **Javascript (JS)** has been kept, then **Javascript** and **JS** would be found in the column *synonym* of the skill's row.
These methods allowed us to delete approximately 3000 duplicates. The figure below is a look at the dataset at this step of the pre-processing :

	_id	name	topSkills	synonyms
0	5b057b59b5d2691588302c95	{'name': 'A + Certified', 'clean_name': 'a + c...	[{'name': 'Certification Network+ ', 'memberCo...	NaN
1	5b057b59b5d2691588302c96	{'name': 'A&d', 'clean_name': 'a&d', 'tokenize...	[{'name': 'Pétrole ', 'memberCount': 694, 'tok...	NaN
2	5b057b59b5d2691588302c97	{'name': 'A&e', 'clean_name': 'a&e', 'tokenize...	[{'name': 'Services de santé ', 'memberCount':...	NaN
3	5b057b59b5d2691588302c99	{'name': 'A&p', 'clean_name': 'a&p', 'tokenize...	[{'name': 'Aviation ', 'memberCount': 6175, 't...	NaN
4	5b057b59b5d2691588302c9a	{'name': 'A&r Administration', 'clean_name': '...	[{'name': 'Musique ', 'memberCount': 67631, 't...	NaN
...	...	...	...	...
30970	5b0bed10d24d6b2a60d27cd6	{'name': '837i', 'clean_name': '837i', 'tokeni...	[{'name': '837P ', 'memberCount': 494, 'tokeni...	NaN
30971	5b0bed10d24d6b2a60d27cd7	{'name': '837p', 'clean_name': '837p', 'tokeni...	[{'name': '837l ', 'memberCount': 494, 'tokeni...	NaN
30972	5b0bed10d24d6b2a60d27cd9	{'name': '8d Problem Solving', 'clean_name': '...	[{'name': 'Failure Mode and Effects Analysis (...	NaN
30973	5b0bed10d24d6b2a60d27ce2	{'name': '960 Grid System', 'clean_name': '960...	[{'name': 'Cascading Style Sheets (CSS) ', 'me...	NaN
30974	5b0bed10d24d6b2a60d27ce3	{'name': '@task', 'clean_name': '@task', 'toke...	[{'name': 'Project Management ', 'memberCount'...	NaN

30975 rows × 4 columns

Figure 6: Dataset after including synonyms

2.2.6 Adding *id* to each elements in *topSkills* (A)

A laborious work has been to browse the entire dataset to complete each element of topskills with its correspondent skill's *id*. We used *clean names* and only searched in skills with the same first letter (the dataset being ordered alphabetically by *name*). After a few hours of computation, we observed that only 80% of the skills in *topSkills* have a correspondent skill in the column *name* and hence have *id*. For instance, **amazon web services** can be found in **aws**'s *topSkills* but can't be found in *name*. Unfortunately, we couldn't create ids as it is a part of xLearn's software which we don't have access to. Here is an example of a skill in a list of *topSkills* :

```
{'_id': '5b057b59b5d2691588307da1',
 'clean_name': 'microsoft office',
 'memberCount': 7278,
 'name': 'Microsoft Office ',
 'tokenized_skill': ['microsoft', 'office'],
 'tokenized_stem_skill': ['microsoft', 'offic']}
```

Figure 7: Example of a skill in a list of *topSkills*

2.2.7 Using wikipedia pages (A)

xLearn suggested a few papers in which we can find ways to delete duplicates. One of papers is "LinkedIn Skills: Large-Scale Topic Extraction and Inference, by Bastian and al" which suggests to use the wikipedia pages. Indeed, if two skills *name* redirect to the same wikipedia page, we consider that they are duplicates. With this method, we achieved eliminating 673 duplicated which are difficult to find such as : **Child and adolescent psychiatry** and **Pediatric psychiatry**.

Finally, we put deleted skills in the *synonyms* columns.

3 Word Embedding Modeling

We had the idea of testing some well known models in the community of Natural Language Processing (NLP). Then, we will test the coherence between skills and see which model provides a better result in our case.

3.1 Word2Vec (A)

One of the most commonly used models to create word embedding is Word2Vec from Google. We decided to give it a try and use every skill concatenated with its *topSkills* to create our sentences. Then, Word2Vec creates a vocabulary and gives each skill a vector of dimension 200 which we added to the database. In order to check if it worked fine, we try some simple operations on word embedding and plot the embedding vector using PCA to represent some distances or clusters.

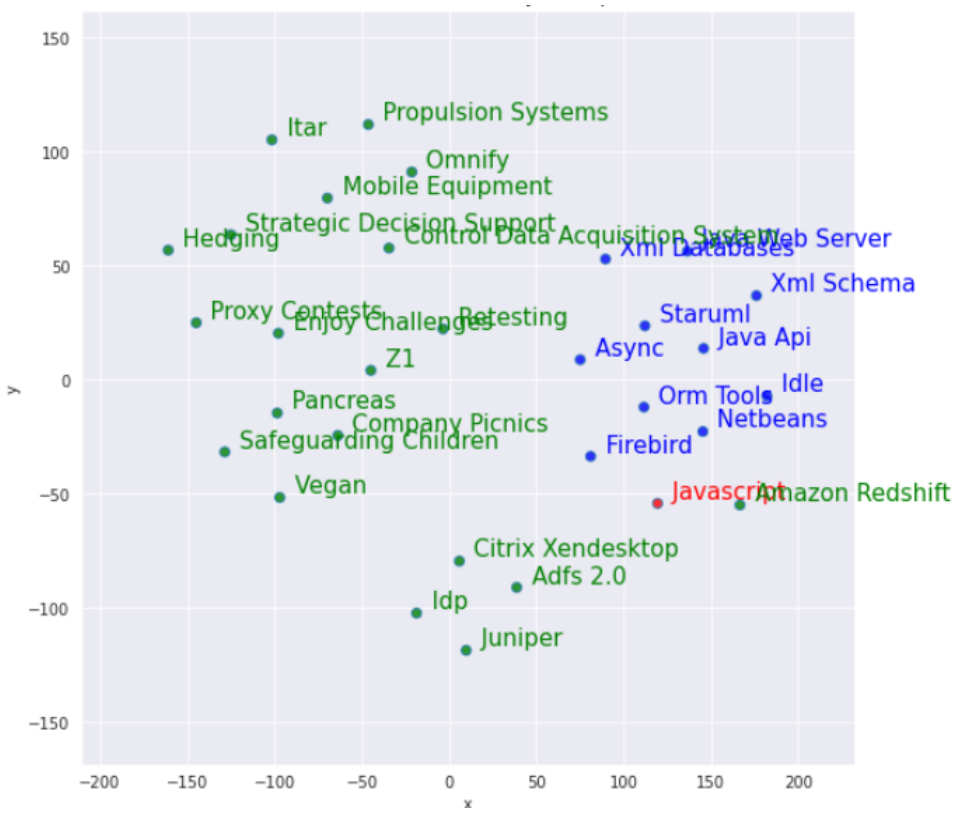

Figure 8: Visualisation of random skills close to **JavaScript**. Skills appearing blue are **JavaScript**'s *topskills*

3.2 FastText (V)

FastText is a model introduced by Facebook. This model is based on the idea of using skipgram model, where each word is represented as a bag of character n-grams. Then, this model generates a vector representation which is associated to each character n-grams and the vector representation of the word is the sum of the n-grams representations corresponding

to the word. One of the main advantages of this model is that it allows users to compute any word representation, even for the words that did not appear in the training set.

We followed the same steps as in the case of Word2Vec in building vocabulary and defining the dimension of vector representation as 200. The following figure is an example of plotting similarity between skills.

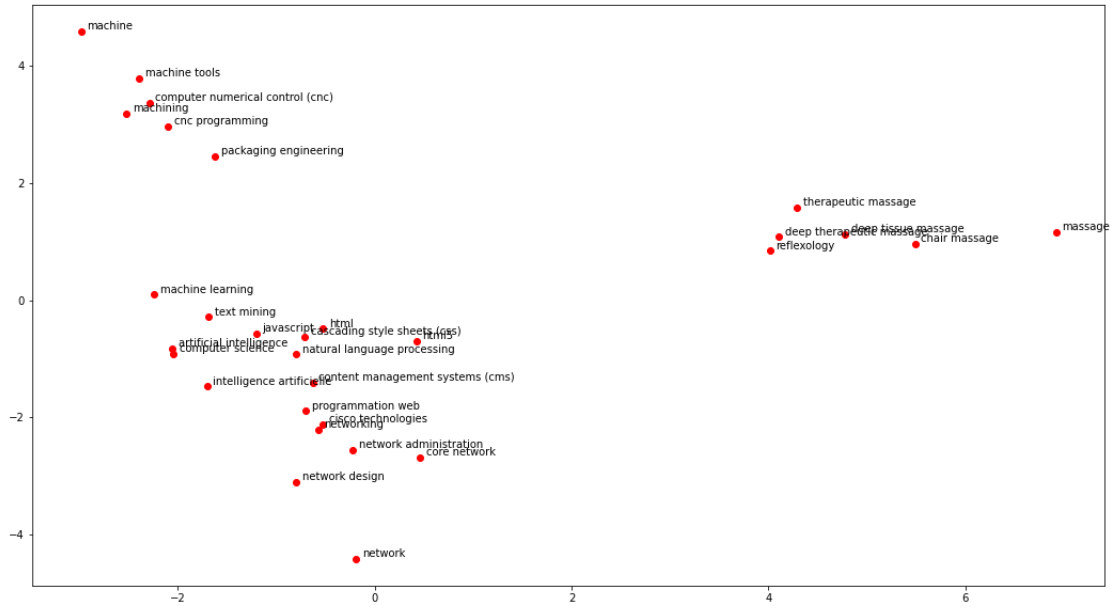

Figure 9: Visualisation of skills that are close to **artificial intelligence**, **html**, **machine**, **network**, and **massage**.

We plot the figure 8 and 9 using Principal Component Analysis (PCA) implemented by sklearn. We observe that both Word2Vec and FastText model make sense in terms of similarity between skills. In figure 9, we see that **machine** is around other skills which is in the same category. The same things happen to **artificial intelligence, html** and **network**. They are skills related to coding. Hence, it is normal to see them in the same cluster. Finally, **massage** is a skill which is completely different compared to other two categories. That is why it is located far away from them, but with other skills similar to it.

3.3 Model decision

It is very difficult to compare two word embedding models for many reasons. One of them is that word similarity is subjective and hard to interpret. Therefore, we decided to go with Word2Vec as it is more widely used than any model. At this point of the project, we have a clean database containing Word2Vec embedding vector for each skill. We are now looking for a way to extract skills from resumes or job descriptions in order to compute their similarity.

4 Methods analysis on skill extraction

xLearn suggested several methods to extract skills from texts: Watson by IBM, Amazon Web Service (AWS), CSO and Skill-ML. Unfortunately, some of these methods such as AWS and

Skill-ML are either not free or unable to run. In the following section, we are going to describe two different tools that can be used to extract skills.

4.1 CSO by the Knowledge Media Institute researchers (A)

The most promising classifier we studied is CSO by the Knowledge Media Institute researchers. This classifier was trained on a large-scale ontology of research areas from Computer Science to Linguistics. This ontology has been using an algorithm named Klink-2, on 16 million publications. This algorithm produces an ontology by processing scholarly metadata of publications and external ressources (DBpedia, web pages...). It takes a set of keywords as input and infers the semantic relationship using certain metrics. You can find the pseudocode of the Klink-2 algorithm in this paper : oro.open.ac.uk/62026.

As researchers refined some relationships manually, they have reached 26K topics and 226k semantic relationships (such as *relatedEquivalent*, *contributesTo*...) in the ontology. As for the classifier, it consists of three main components that you can see on figure 10. Firstly, the syntactic module parses the input documents and identifies CSO concepts that are referred in the documents via n-grams and concept similarity. The semantic module uses part-of-speech tagging to identify promising terms. Finally, the CSO Classifier uses the results of both of these two modules and enhances them by adding relevant skills.

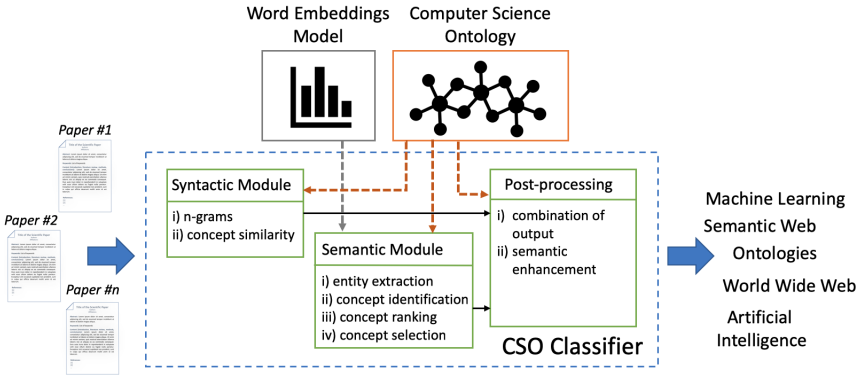

Figure 10: Architecture of the CSO classifier

The main reason why we focused on this classifier is that it is possible to do transfer learning easily by changing their dataset of skill with ours. However, this also requires an ontology of skills inheritance to do so, which we don't have. But this is still promising as such an ontology is feasible.

4.2 Watson natural language understanding by IBM (V)

IBM has a tool that allows users to train their machine learning model which is capable of identifying entities, coreference and relationships of interest. In our case, we are interested in extracting an entities of type skill. The following figure represents the workflow for developing a machine learning model.

This machine learning model is trained in a way of supervised learning. We give a set of documents in order to train the model. Before being able to train the model, we need to annotate each document the entity that we want to extract, for example in our case we need

Figure 11: Workflow for developing a machine learning model on IBM cloud

to annotate all skills in the resumes. All these annotations will be later considered as ground truth. Next, we can train the model by using a proportion of the ground truth. During the training, the model try to capture the pattern of the entity which it needs to extract. After the training process, we get a model along with the evaluation in form of confusion matrix containing *F1, Precision* and *Recall*. One main advantage of this tool of IBM is that we can deploy an intermediate model resulting from small training dataset and add more data to the training dataset in order to improve the performance.

In short, we can say that IBM works like a *black box* : it essentially takes a set of annotated documents as input and return a model trained which is capable of extracting skills from resumes.

5 Skill Extraction and Job Description-Resume Similarity Comparison

5.1 Skill Extraction

5.1.1 CSO by the Knowledge Media Institute researchers (A)

For now, we used the classifier to extract skills according to their dataset in order to create our own method to extract skills. Furthermore, as explained above, we focused only on the enhanced (post-processing) classifier, one of the three classifiers proposed by the Knowledge Media Institute researchers. Here is the method we developed :

- On the one hand, we do a search word-by-word in the job description to find skills which exist in our dataset. We also created an option to do this search only in a 'skill section'.
- On the other hand, we use CSO classifier to extract skills and only keep those that are in our dataset.

We now have two sets of skills retrieved differently, we will call them respectively S and C.

We decide to compute a value V to represent 'how much' a skill is relevantly extracted from the job description. The lower $V(s)$ is, the more it is meaningfully extracted. Here is how it is done for a skill s :

$$V(s) = \begin{cases} 0, & \text{if } s \in S \cap C \\ \lambda_S(\sum_{s' \in S} ||s - s'|| + \alpha \min_{b \in S \cap C} ||s - b||), & \text{if } s \in S \\ \lambda_C(\sum_{s' \in C} ||s - s'|| + \alpha \min_{b \in S \cap C} ||s - b||), & \text{if } s \in C \end{cases}$$

As you can see, there are 3 hyperparameters, which will allow us to explain this formula by steps. First, α represents how important it is that the skill is close to one of the best skills extracted (those two skills found by the CSO classifier and by the word-by-word search). The other part of the formula tells how close the skill is to the others of its set. Then, λ_S and λ_C represent how much we put our trust in the word-by-word search or in the CSO classifier. The lower the value, the more we think it is efficient. Here are the value we chose for the hyperparameters :

hyperparameter	value
α	1.5
λ_S	0.75
λ_C	1

This way, we can sort skills from the lowest to the highest value. Hence, we can consider a threshold where skills are kept in the skill extraction or not.

```
[('software', 0),
 ('ruby', 0),
 ('api', 1.708426959538866),
 ('sdk', 1.710658330462803),
 ('integration', 1.9859672132491726),
 ('gateway', 2.0724513995846086),
 ('class', 2.150636261708377),
 ('clas', 2.2286679330864985),
 ('developer tools', 2.707086380193385)]
```

Figure 12: Example of a skill extraction for a job description about developing a software allowing automatic generation of API documentation

5.1.2 Watson by IBM (V)

We trained our IBM machine learning model with a small resumes dataset found on the internet. Due to the lack of amount of the data, the model is not quit accurate in general. Nevertheless, it is capable of capturing or extract some skills if we give as input a resume from the same domain as training dataset.

In the figure 13, we can see the evaluation of the our IBM machine learning model. Keep in mind that we trained our model on a dataset of 10 resumes and 4 job descriptions. This is the main reason behind the low score in general. Nevertheless, if we look at the result that the model is capable of extracting from the resume of the same domain of dataset in figure

Entity Types	F1	Precision	Recall	% of Total Annotations	% of Corpus Density (by number of words)	% of Documents that Contain This Type
Skill	0.26	0.47	0.18	100% (100/100)	5% (100/1865)	100% (3/3)

Figure 13: Score of IBM machine learning model

14, it does pretty well and extracts almost all the skills and work experience of the candidate which are important in order to compute the similarity with job description.

```
['visio',
 'access',
 'powerpoint',
 'excel',
 'professional skills',
 'personal assistant',
 'night secretary',
 'ms office',
 'words',
 'executive assistant',
 'administrative management',
 'power point',
 'managing director',
 'marketing activities',
 'diploma management studies',
 'personal assistant',
 'senior secretary',
 'excel']
```

Figure 14: Skills extracted using IBM model

5.2 Job Description-Resume Similarity Comparison

5.2.1 Method based on CSO classifier (A)

Now that we can extract skills into a sorted list, we want to compare a job description and a resume according to their respective list of skills extracted L_j and L_r. As euclidean distances were useful to sort skills, we need a meaningful comparison metric. We will use the cosine similarity. Here is the formula for the similarity between two lists of skills, one from a job description and another from a resume :

$$S(L_j, L_r) = \frac{1}{Card(L_j)} \sum_{s_j \in L_j} \max_{s_r \in L_r} \frac{s_j \cdot s_r}{|s_j||s_r|}$$

For each skill of the job description, we are looking for the closest skill in the resume. If the skill is in the resume, then the max will be 1. Finally, if every skill of the job description is in the resume, the sum will be equal $Card(L_j)$ and the similarity between the two documents will be 1.

5.2.2 Method based on FastText model (V)

By loading the FastText model trained above on our database of skill, we are capable of calculating the similarity between a resume and a job description. Once again, the result is based on the capability of machine learning model from IBM. After extracting skills from both resume and job description, we have two lists of skills. FastText has a function called *n_similarity* which takes two lists of skills and calculates the cosine similarity between them. We are going to give an example of results obtained from the IBM model with FastText.

We compare the similarity between a job description and 2 candidates' resume. The job description is looking for an *assistant manger*. The first candidate has experience in *management* and the second candidate has experience as a *legal counselor*. The result is interesting because the similarity between job description and the first candidate goes up to 0.88 while the similarity between job description and the second candidate is at 0.47 which makes sense in our use case since the profile of an assistant manager is more similar to the profile of someone having experience in management than to a legal counselor.

6 Implementation

6.1 Introduction to the implementation

One of the results of this project was the cleaned database, but also to create an API for xLearn to use. We suggested that a web interface could also be interesting to produce. In order to host this API and web interface, we used *pythonanywhere*. This website allows to freely host a web application using *Flask*, a framework in python for web development. The following part will explain how this app works. You can find it at : *pfeprimxlearn.pythonanywhere.com*

6.2 The API (A)

Basically, an API is a web page that users interact with using requests. With the data provided by the users, the API executes a set of operations and then returns the result as a JSON.

Our API works using only POST requests. These requests must contain a *title* field and an *abstract* field, representing respectively the title and the content of the document. The *keywords* and *threshold* fields are optional. This *threshold* is the value of distance above which the program won't keep the skills. Of course, you can tweak the hyperparameters with the optional fields *lambdaC*, *lambdaS* and *alpha*. Finally, it is possible to disable the search for a 'skill section' by giving $skill\_section = False$, which has a great impact on the results.

Here is an example in python :

```
import requests
job_description = {"title": "...", "abstract": "..."} #TO FILL
r = requests.post("https://pfeprimxlearn.pythonanywhere.com/api/", data=job_description)
print(r.text)
```

The API returns a JSON containing the skills extracted ordered by relevance using the algorithm associated to the CSO classifier, as previously explained.

6.3 The web interface (A)

The web interface is hosted on the same website as the API. It has two pages (cf Annexes). The first one (figure 15) allows users to submit a title and a description of a job description to extract skills using our method from our API. We suggested an example found on Indeed London. The second page (figure 16) grants the computation of the Job description-Resume similarity as explained above. However, the formula has been changed into the following :

$$S(L_j, L_r) = \frac{1}{Card(L_j)} \sum_{s_j \in L_j} \max_{s_r \in L_r} \frac{s_j \cdot s_r}{|s_j||s_r|} \max\left(log(10 - V(s_j)), 0\right)$$

Indeed, for more relevant results, we decided to keep track of the value of the skills extracted. We want that skills with low values have greater impact on the similarity and vice versa. So we chose the function $f(x) = log(10 - x)$ arbitrarily because it corresponds to our needs : a function slowly decreasing with $f(0) = 1$.

We created a resume that we believe fits good with the job description, which gets a similarity of 0.92.

7 Conclusion

The real data is rather complicated to deal with. We spent a lot of time on trying to figure out what to do and to accomplish the task, for instance the pre-processing step. Some of the important operations can take up to hours or days to execute. Therefore, one of the main challenges we faced is the necessity to adapt with problems and the resources that we possess in hand: time, data, tools (free plan of IBM with limited number of services, free access to Google Cloud for skills translation with limited time), CPU and GPU (we did the whole project on Google Colab). Nevertheless, thanks to our supervisor from xLearn, M. Mohammad Fallaha, and our hard-working, we are glad that we are able to demonstrate the work done with this report and moreover, with a web interface and API for users to interact with.

In order to conclude, this work can extended in these directions:

- Our web interface is implemented based on Word2Vec model as the comparison between two word embedding models (in our case Word2Vec and FastText) is complicated which involves humans responses. This comparison is needed to be done if the resource allows.
- We can create the ontology of skills needed for CSO to work with our database. Nevertheless, we would require more relative data in this case.
- The web interface can be developed in a way that it allows users to upload many resumes and find the one fitting the most with the job description. For current version, it can handle only one resume in the form of plain text.
- We can add more data to train the model of IBM in order to improve its performance.

Annexes

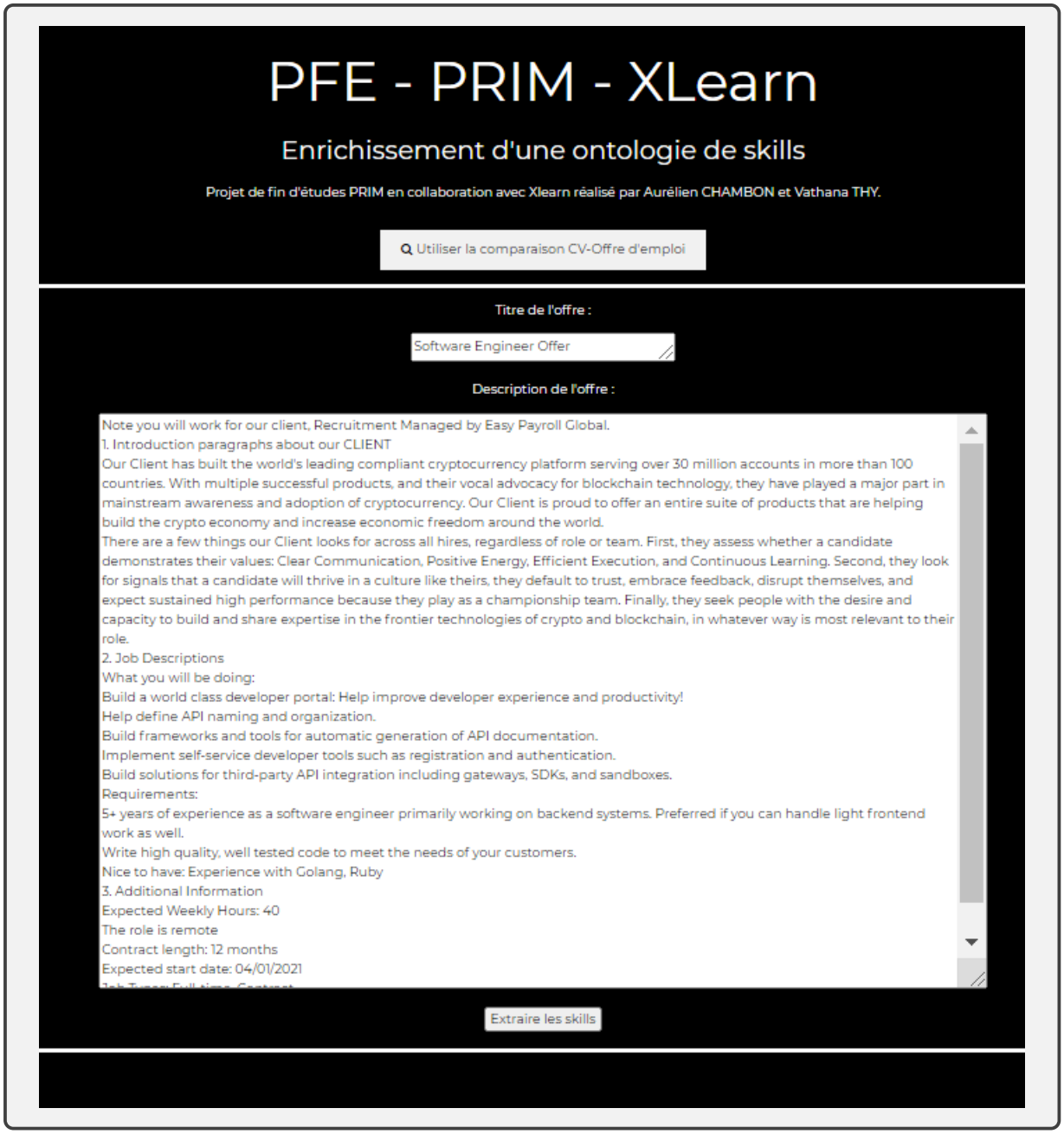

Figure 15: First page of the two pages of our web interface, focused on skill extraction

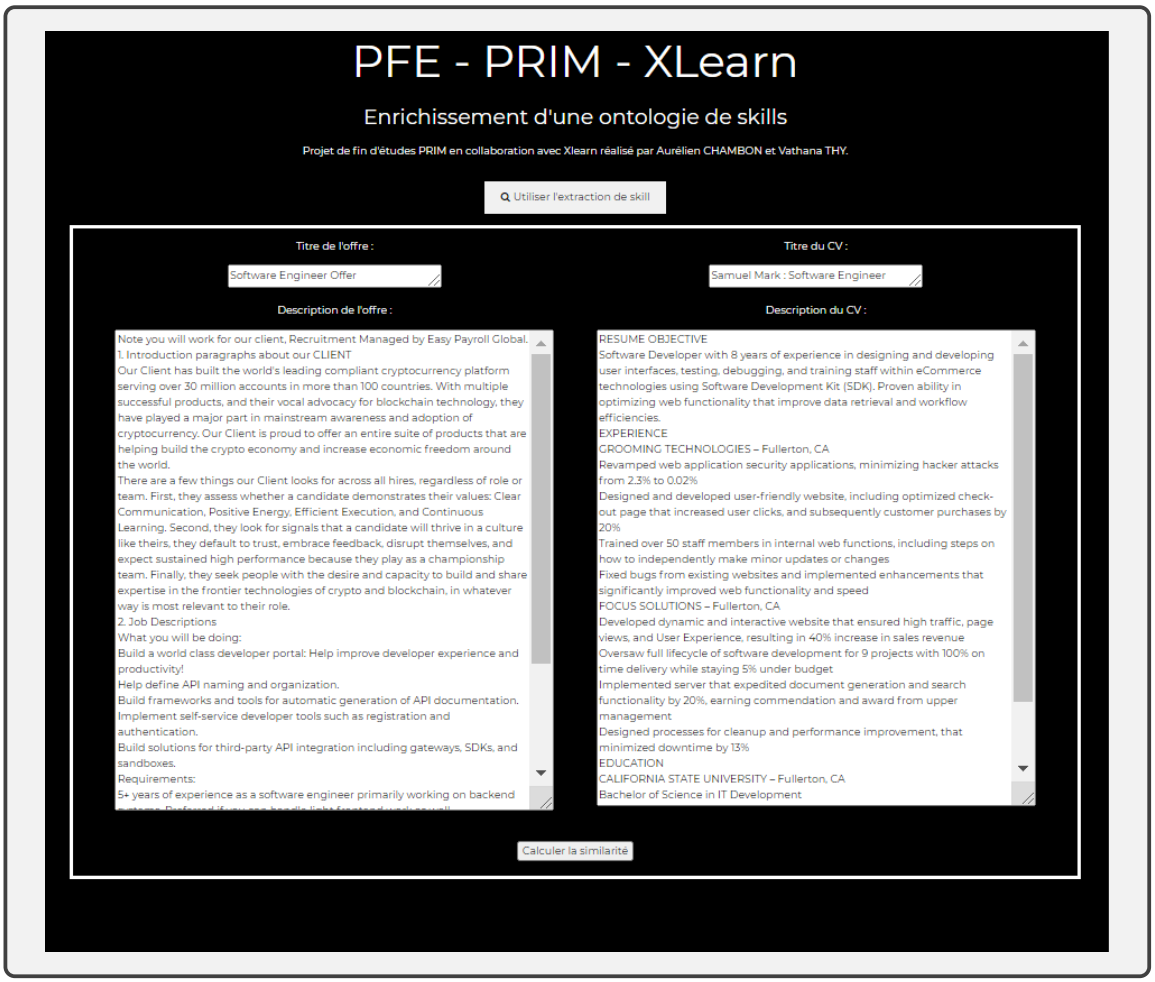

Figure 16: The Second page, allowing the computation of Job description-Resume similarity